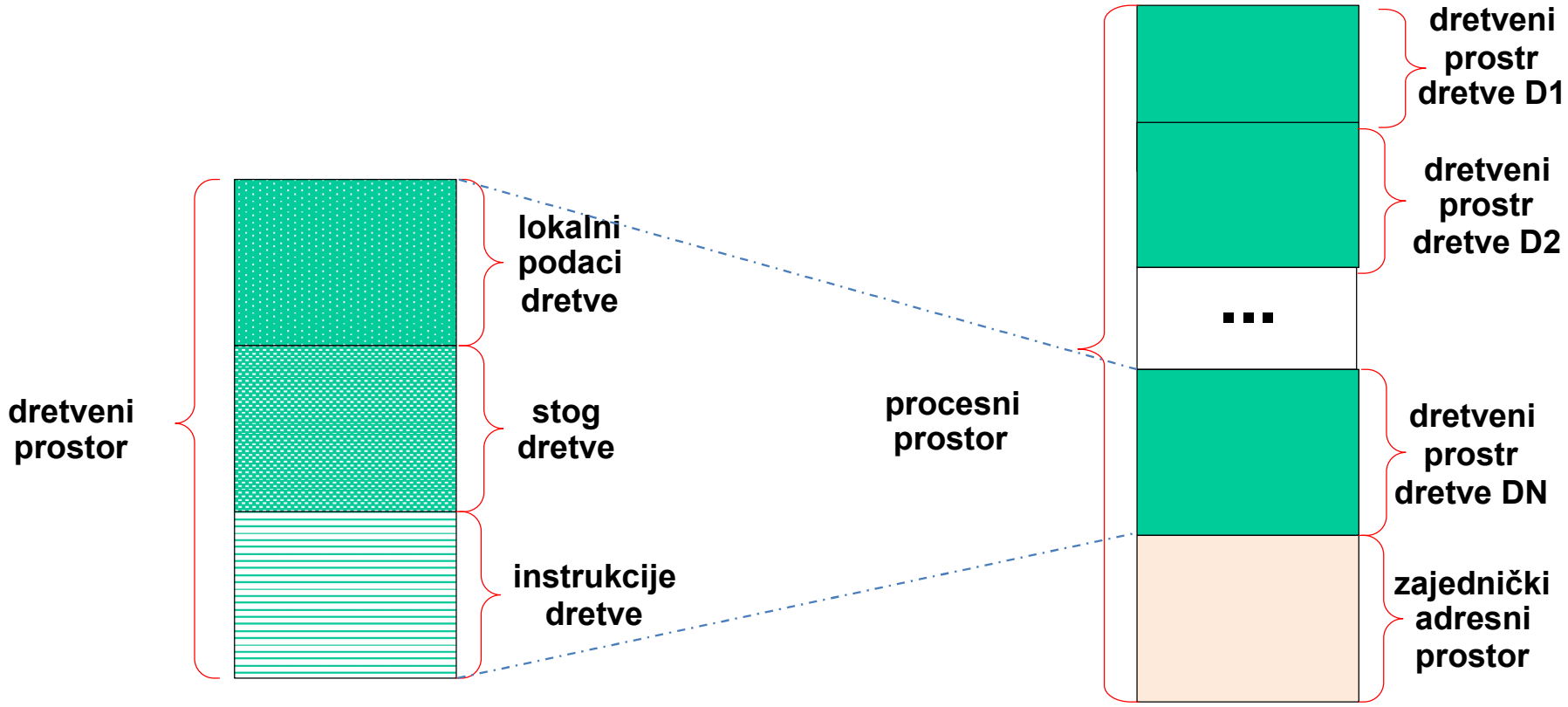
1. Komunikacija između procesa

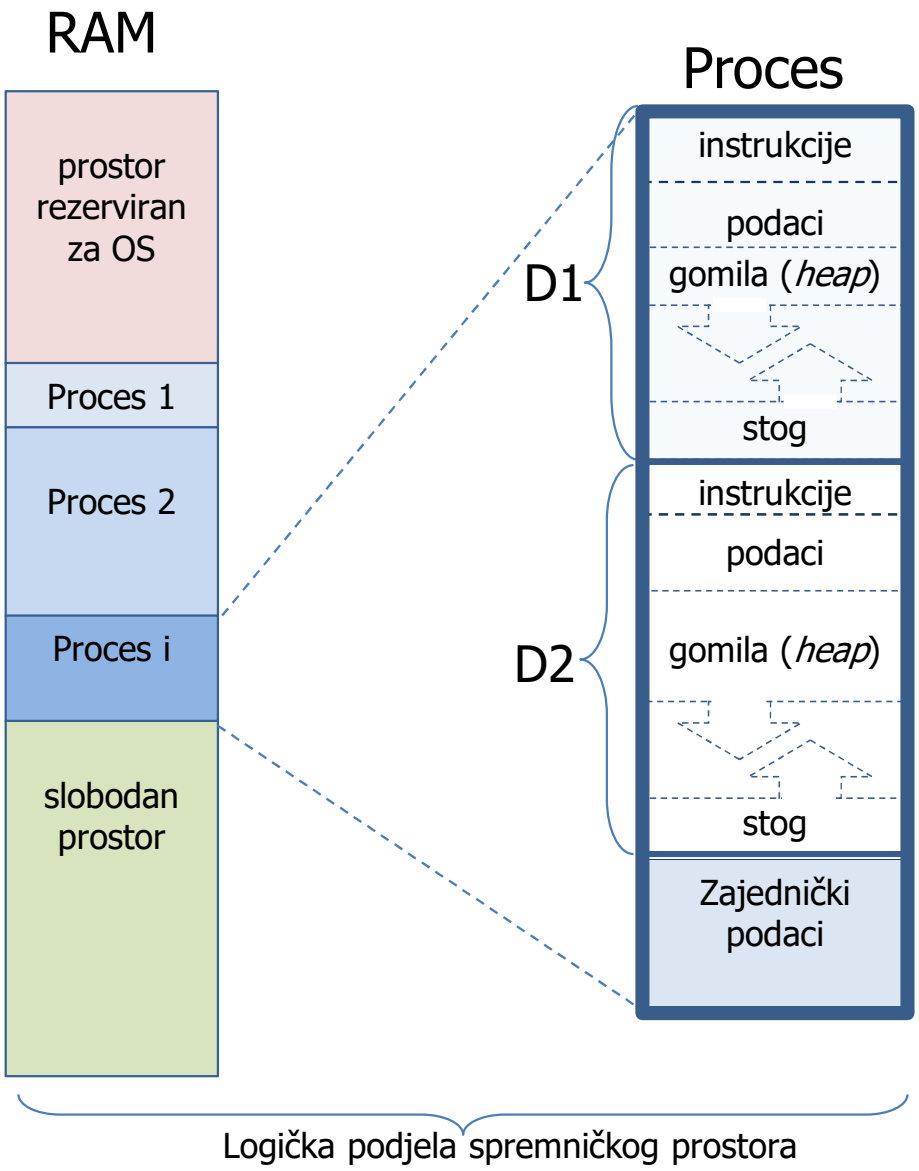
Deseto poglavlje u udžbeniku L. Budin, M. Golub, D. Jakobović, L. Jelenković, Operacijski sustavi

1.1. Komunikacija između procesa unutar jednog računalnog sustava

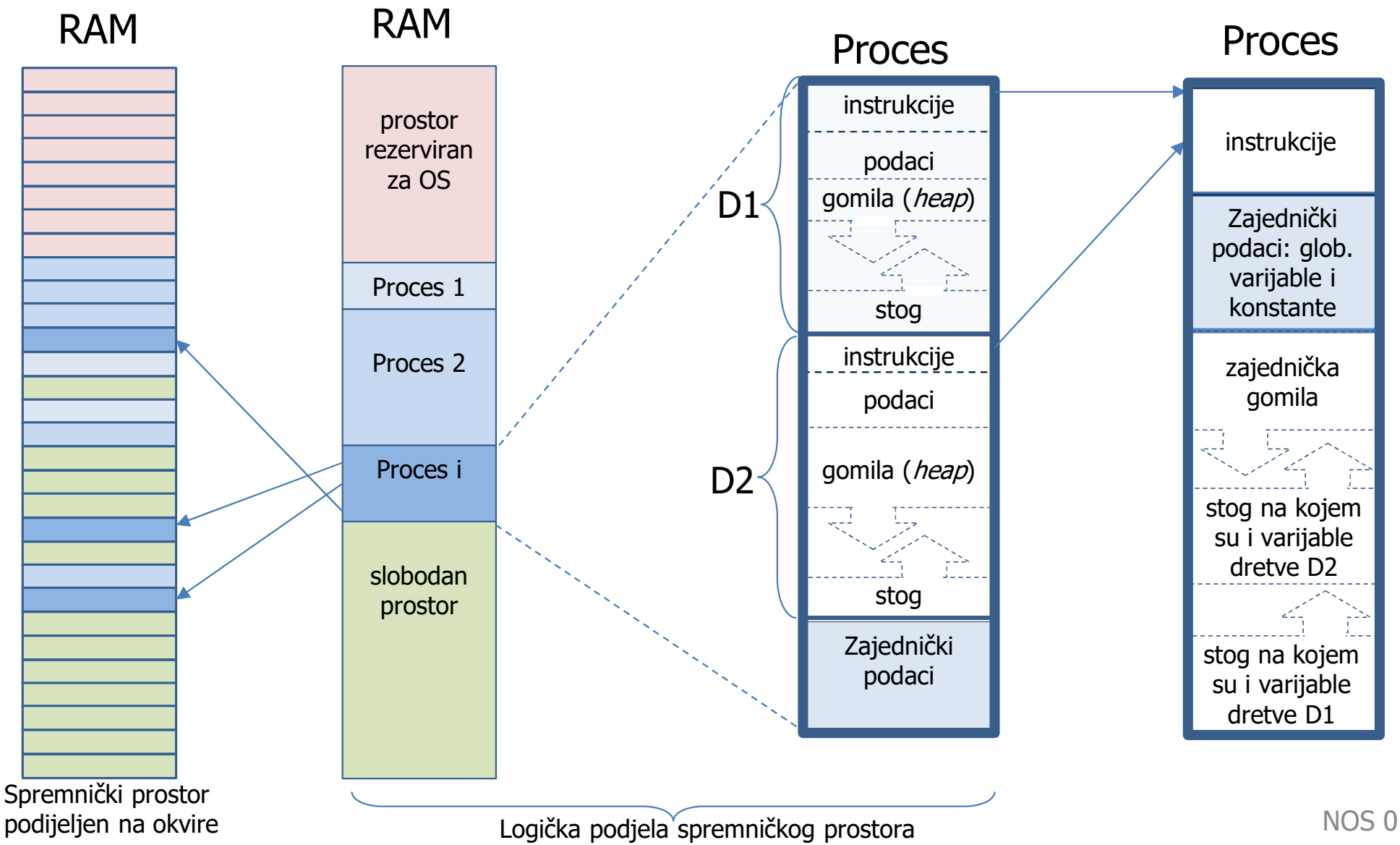
- sve dretve jednog procesa djeluju u njima zajedničkom adresnom prostoru procesa



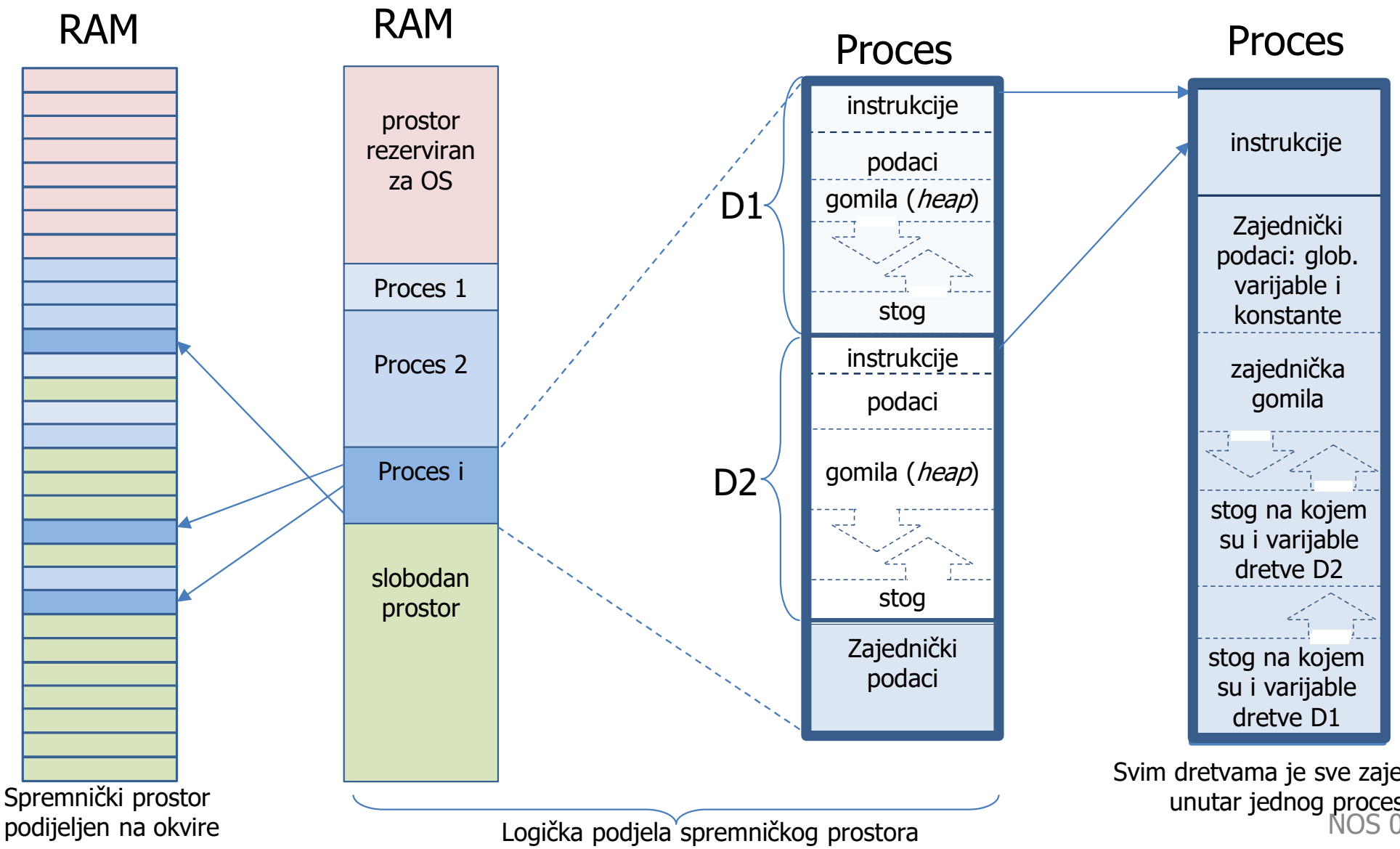
Ponavljanje: organizacija spremničkog prostora?



Ponavljanje: organizacija spremničkog prostora?



Ponavljanje: organizacija spremničkog prostora?



Komunikacija između dretvi istog procesa

- zajednički adresni prostor = dijeljeni adresni prostor
- iz svake dretve može se pohranjivati i dohvaćati sadržaj svih lokacija procesa
- i u takvim uvjetima međudretvena komunikacija se nastoji ostvarivati na nekim sustavnim mehanizmima (komunikacija između proizvođača i potrošača koristeći međuspremnik)

Komunikacija između procesa
tj. komunikacija između dretvi različitih procesa

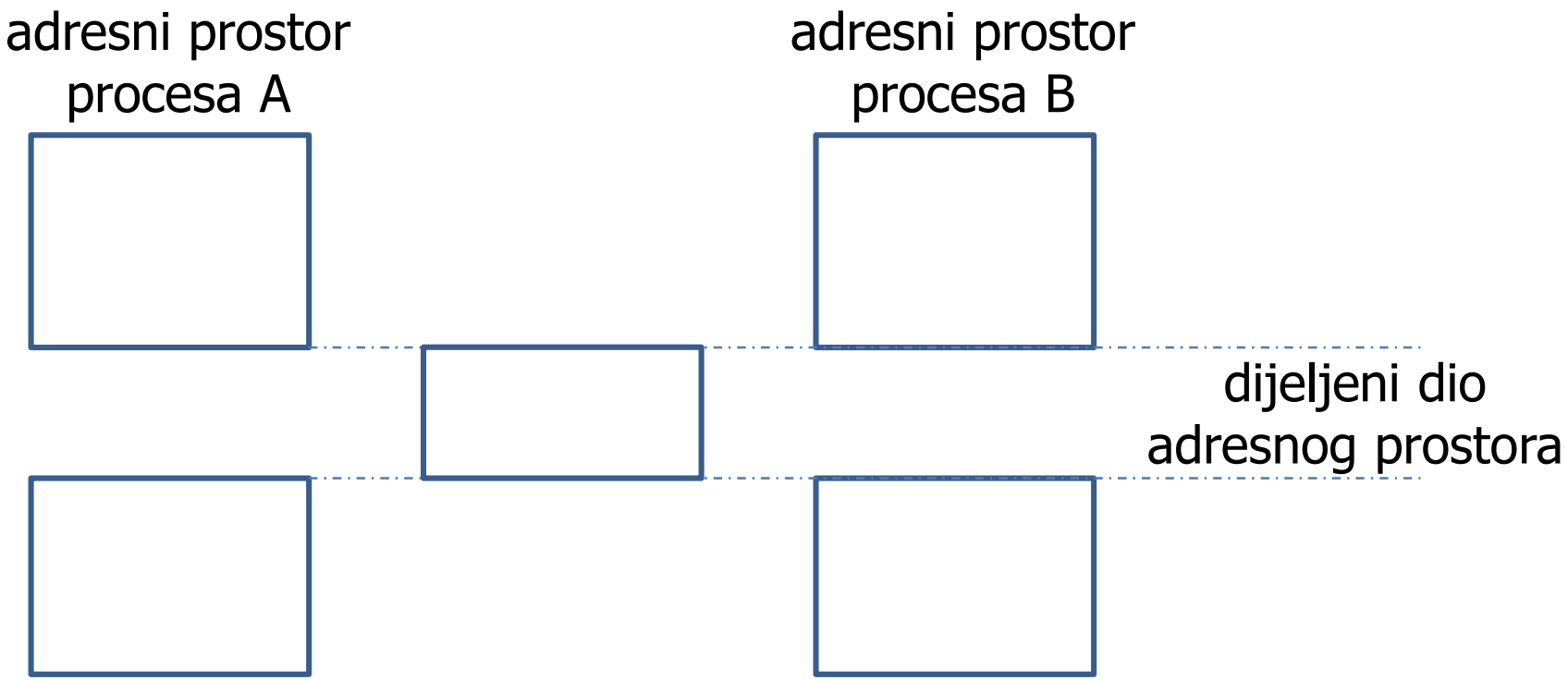
Procesi imaju namjerno razdvojene adresne prostore kako jedan drugome ne bi smetali.

Načini komuniciranja između procesa unutar nekog (jednog) računalnog sustava:

- uz pomoć dijeljenog spremničkog prostora
- razmjenom datoteka (najjednostavniji način)
  - proizvođač proizvede datoteku i pohrani je na disk
  - potrošač otvori tu datoteku i preuzme njezin sadržaj
- uz pomoć varijabli okoline
- razmjenom poruka
- cjevovodima

Dijeljeni spremnički prostor

- dio virtualnog adresnog prostora dva ili više procesa može se proglasiti zajedničkim prostorom
- primjer iz OS labosa: `shmat, shmget, shmdt`



Primjer 10.1.

U zbirci Win32 API nalaze se funkcije koje omogućuju da se cijele datoteke preslikaju virtualni adresni prostor procesa i zatim ju dijele s drugim procesima: `CreateFileMapping()`, `MapViewOfFile()`, `MapViewFileEx()`, `OpenFileMaping()`.

- dijeljenje adresnog prostora svodi na dinamičko dijeljenje datoteke preslikane u virtualni adresni prostor procesa
- datoteka koja je preslikana u adresni prostor naziva se preslika datoteke (engl. *file view*).
- funkcija `MapViewOfFileEx()` čak dozvoljava da se preslika datoteke smjesti u željeni dio adresnog prostora

Razmjena poruka između procesa

- u monitorskom načinu ostvarenje problema proizvođača i potrošača razmotren je u odjeljku 6.3. dvije monitorske funkcije:

```
poslati_poruku(p);

prihvatiti_poruku(r);
```

- strukturu podataka potrebnu za ostvarenje tih funkcija te međuspremnik za razmjenu poruka smjestili smo u adresni prostor procesa
- na sličan bi se način mogle izgraditi API funkcije za komunikaciju između procesa, ali ne u adresnom prostoru procesa već u adresnom prostoru operacijskog sustava

Redovi poruka
i
Cjevovodi

Priprema za vježbe

Protokolni slog

- o tehničkom aspektu Interneta vodi brigu konzorcij *Internet Engineering Task Force (IETF)*
  - IETF je utrdio protokolni slog koji se danas pretežito upotrebljava pri povezivanju računala u mrežu
  - taj protokolni slog ima 4 razine:

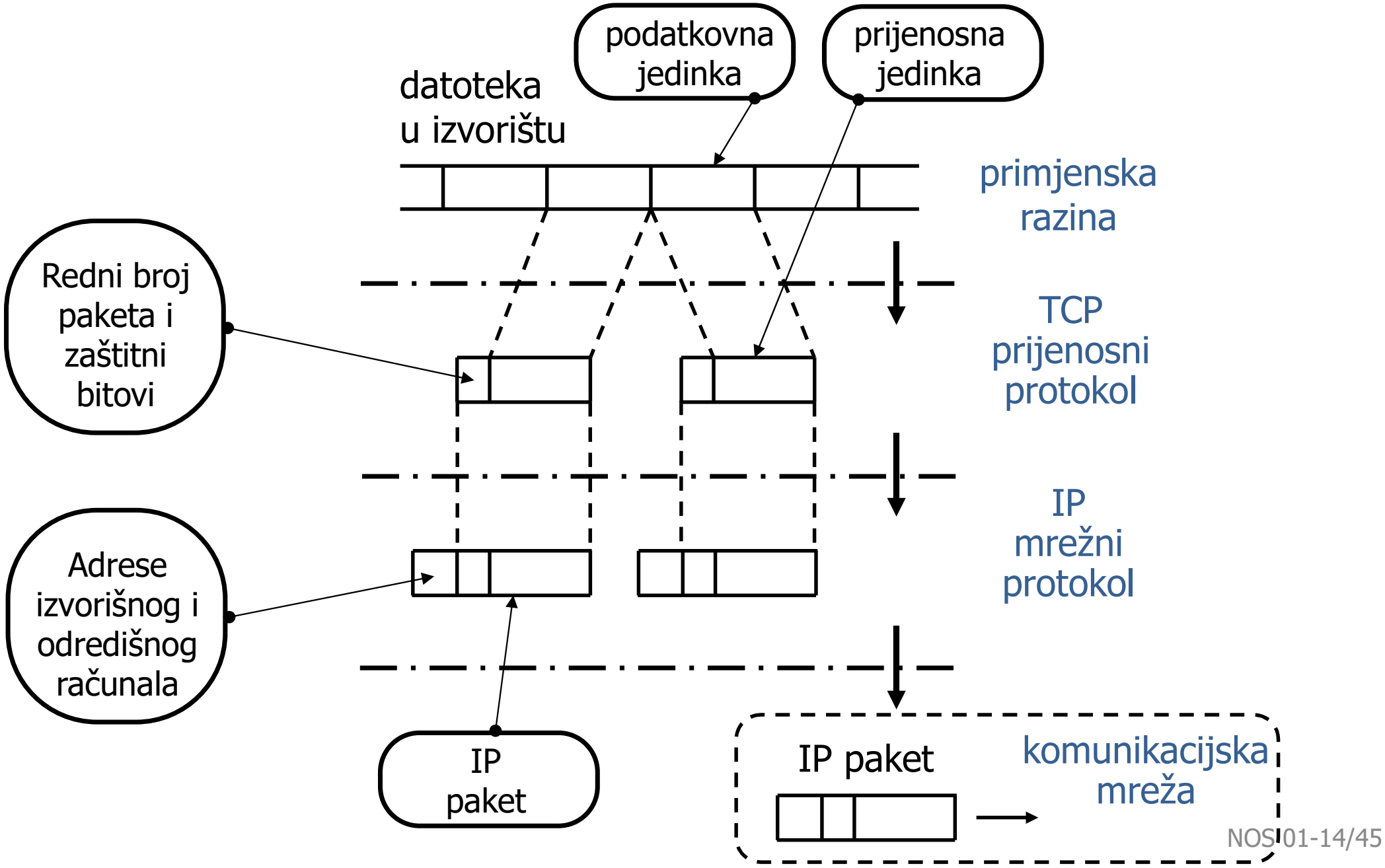
1. primjenska razina (*application layer*)
2. prijenosni protokol (*Transmission Control Protocol* - TCP)
   - određuje kako se datoteka dijeli na dijelove jednake veličine
   - dodaje redni broj i zaštitine bajtove

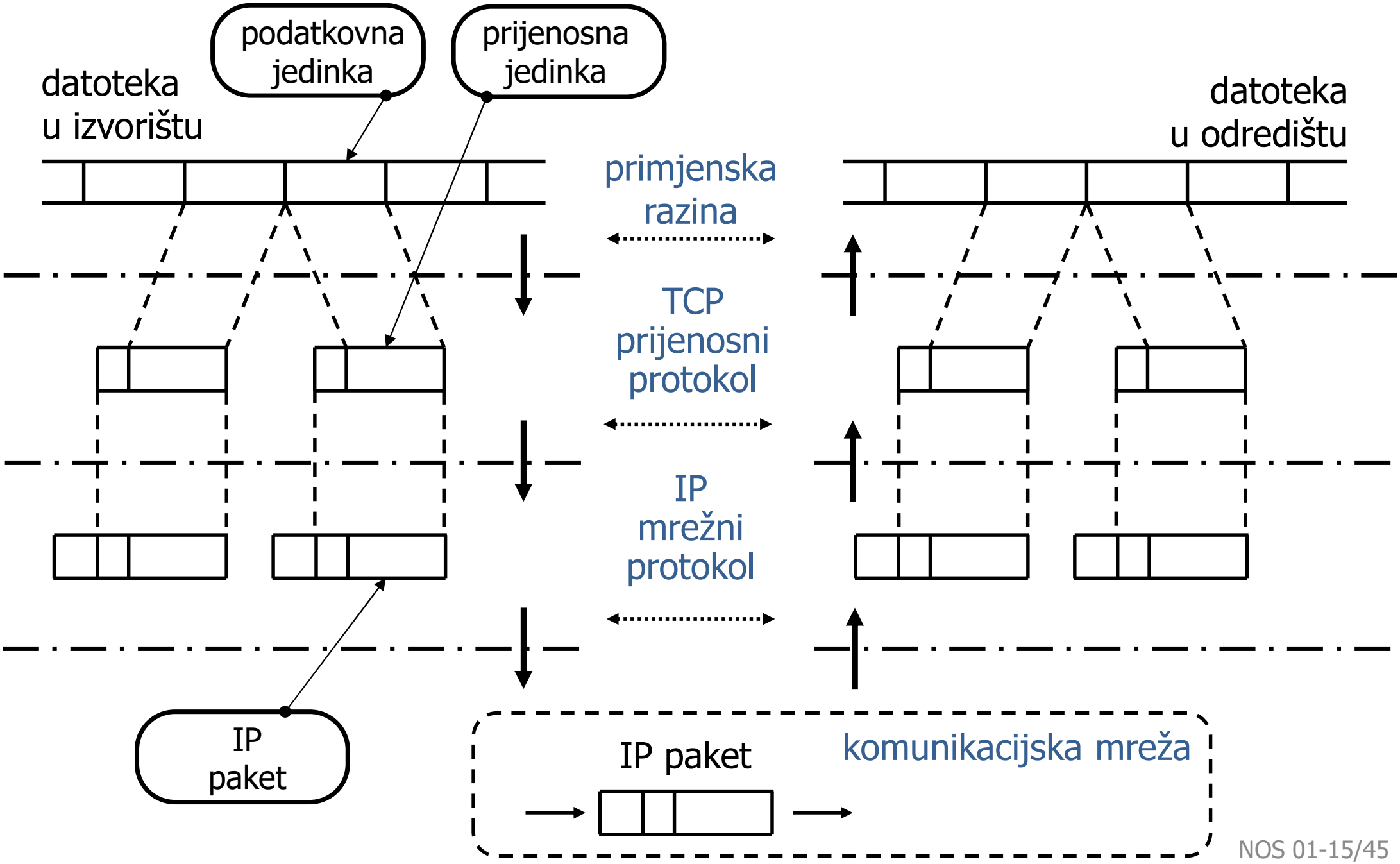
3. IP (od *Internet Protocol*) dodaje zaglavlje unutar kojeg se nalaze adrese izvorišnog računala (*source*) i odredišnog računala (*destination*) - stvara *IP* paket

   "Ako je Internet knjiga, tada su *IP* – stranice, a *TCP* jezik na kojem je knjiga napisana."

4. fizička razina - ona nije posebice normirana

- velik je uspjeh *TCP/IP* polučio je upravo zbog toga što je prijenos paketa moguć uporabom različitih mrežnih tehnoloških rješenja


podatkovna jedinka
prijenosna jedinka
datoteka u izvorištu
primjenska razina
Redni broj paketa i zaštitni bitovi
TCP prijenosni protokol
IP mrežni protokol
Adrese izvorišnog i odredišnog računala
IP paket
IP paket
komunikacijska mreža


podatkovna jedinka
prijenosna jedinka
datoteka u izvorištu
datoteka u odredištu
primjenska razina
TCP prijenosni protokol
IP mrežni protokol
IP paket
IP paket
komunikacijska mreža

1.2. Komunikacija između procesa u raspodijeljenim sustavima

Komunikacija razmjenom poruka

- osnovni model komuniciranja
- u primjenskoj razini (iznad *TCP/IP*) izgrađeni su komunikacijski mehanizmi za uspostavljanje protokola razmjene poruka
- komunikacijski mehanizmi aktiviraju se odgovarajućim API funkcijama
- poruke se oblikuju kao *IP* paketi
- kada proces želi komunicirati on mora uspostaviti komunikacijsku priključnicu ili pristupnu točku (*socket*)

- funkcijom

  `uspostaviti_priključnicu (identifikator,tip,protokol);`

  proces uspostavlja vezu s prijenosnom razinom unutar svog računala kroz jednu pristupnu točku (socket)

- uspostavljena pristupna točka se mora imenovati (povezati s imenom):

  `povezati (identifikator, adresa);`

- ako dva računala uspostave kompatibilne pristupne točke, onda ih se može međusobno povezati asimetričnom funkcijom

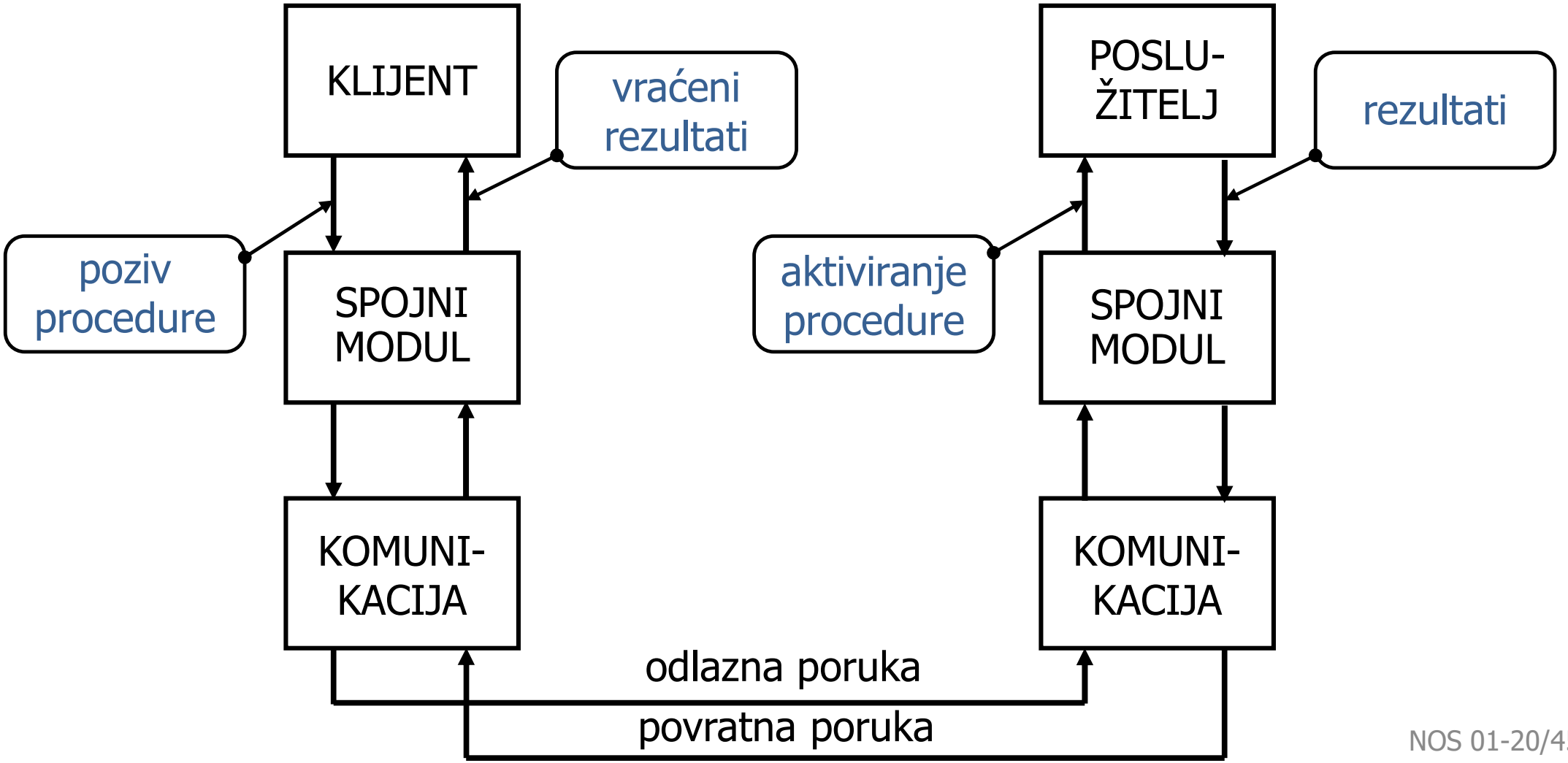
  `povezati (identifikator, adr_vlastita, adr_partnera);`

- ovakav je način uspostavljanja veze prikladan za komuniciranje klijenata i poslužitelja (na sličan način kao kod komuniciranja kroz cjevovod)
- poslužitelj stvara priključnicu, tj. pristupnu točku, povezuje s njom ime i čeka da se neki klijent poveže na nju
- kada posluželj uspostavi vezu tada može početi slanje poruka na sličan način kako se to obavlja preko dvostranog cjevovoda unutar jednog računala

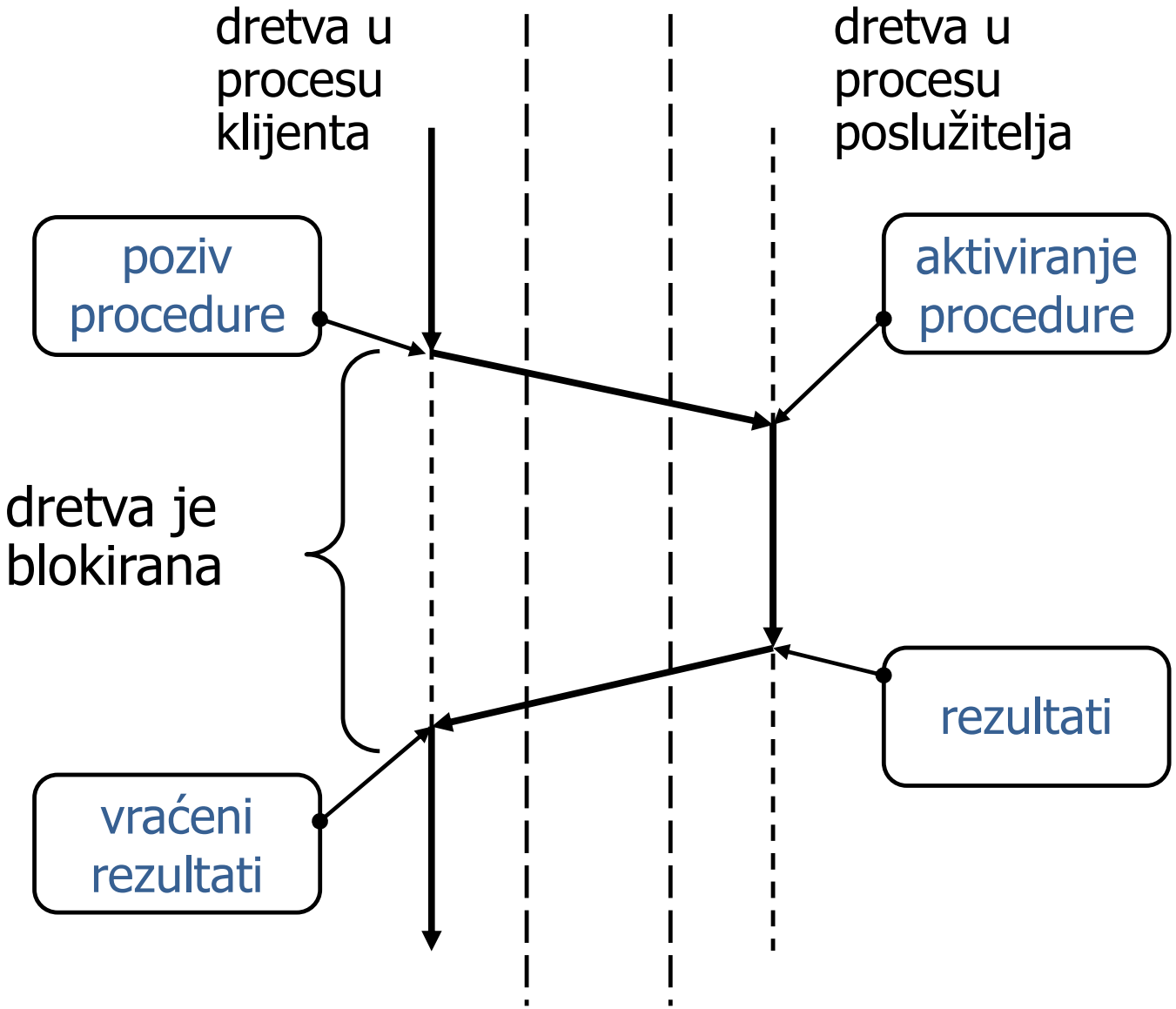
Poziv udaljenih procedura
(*Remote Procedure Call – RPC*)

- svojevrsna nadgradnja mehanizama razmjene poruka
- mehanizam oponaša poziv podprograma koji se pripremaju u adresnom prostoru procesa (poglavlje 2)
- instrukcije procedure se ne nalaze u adresnom prostoru procesa iz kojeg se pozivaju
- razmjena ulaznih podataka i rezultata se obavlja razmjenom poruka prenošenjem vrijednosti (*call by value*)
- poziv podprograma prihvaća spojni modul (engl. *stub*) koji oblikuje poruku i prepušta poruku komunikacijskom sustavu
- poruka se upućuje računalu koje će izvesti proceduru, a rezultate će povratnom porukom vratiti procesu koji je pozvao proceduru

- proces koji poziva proceduru je ustvari klijent, a proces koji izvodi proceduru je poslužitelj

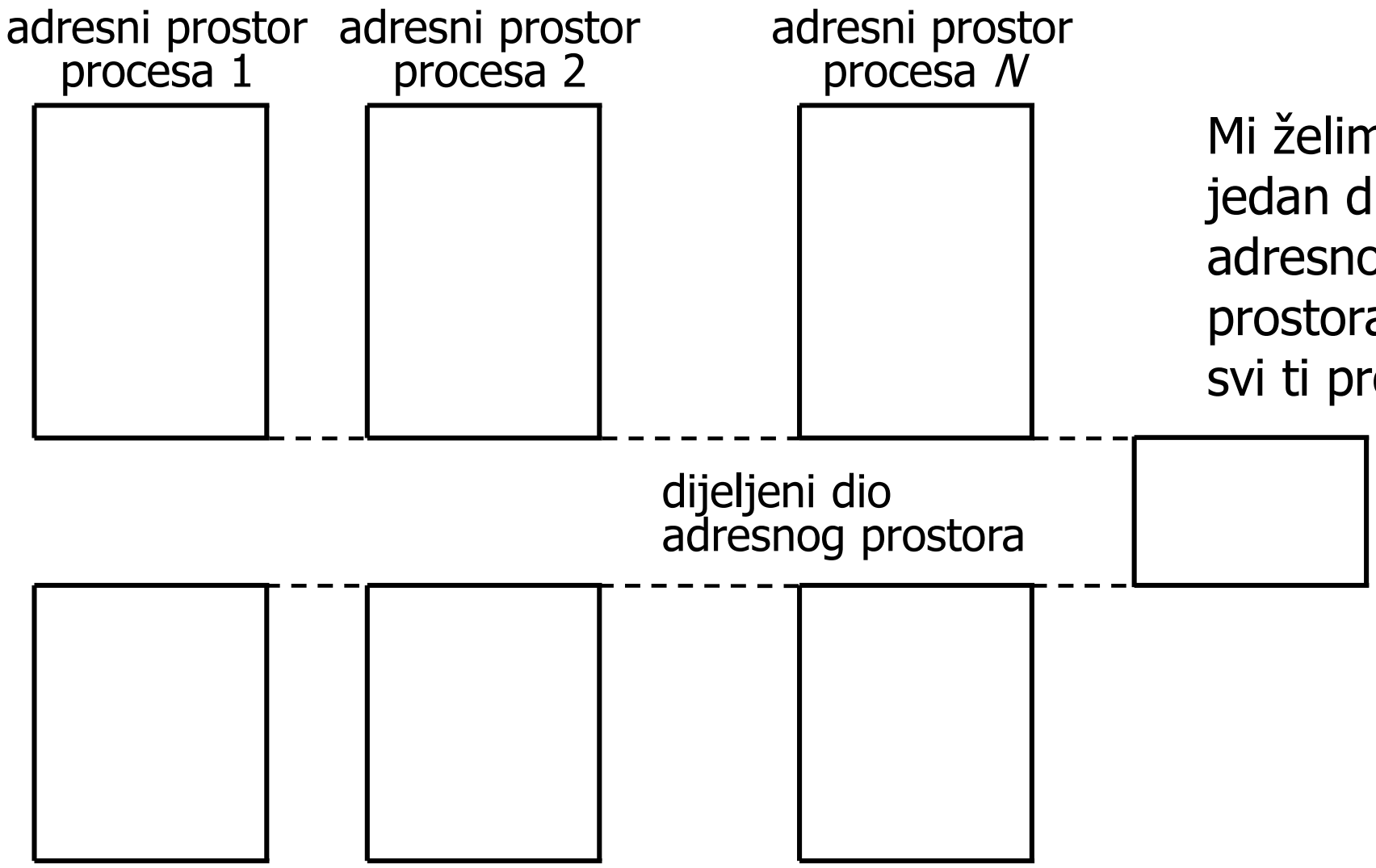


Najjednostavniji protokol poziva udaljenih procedura



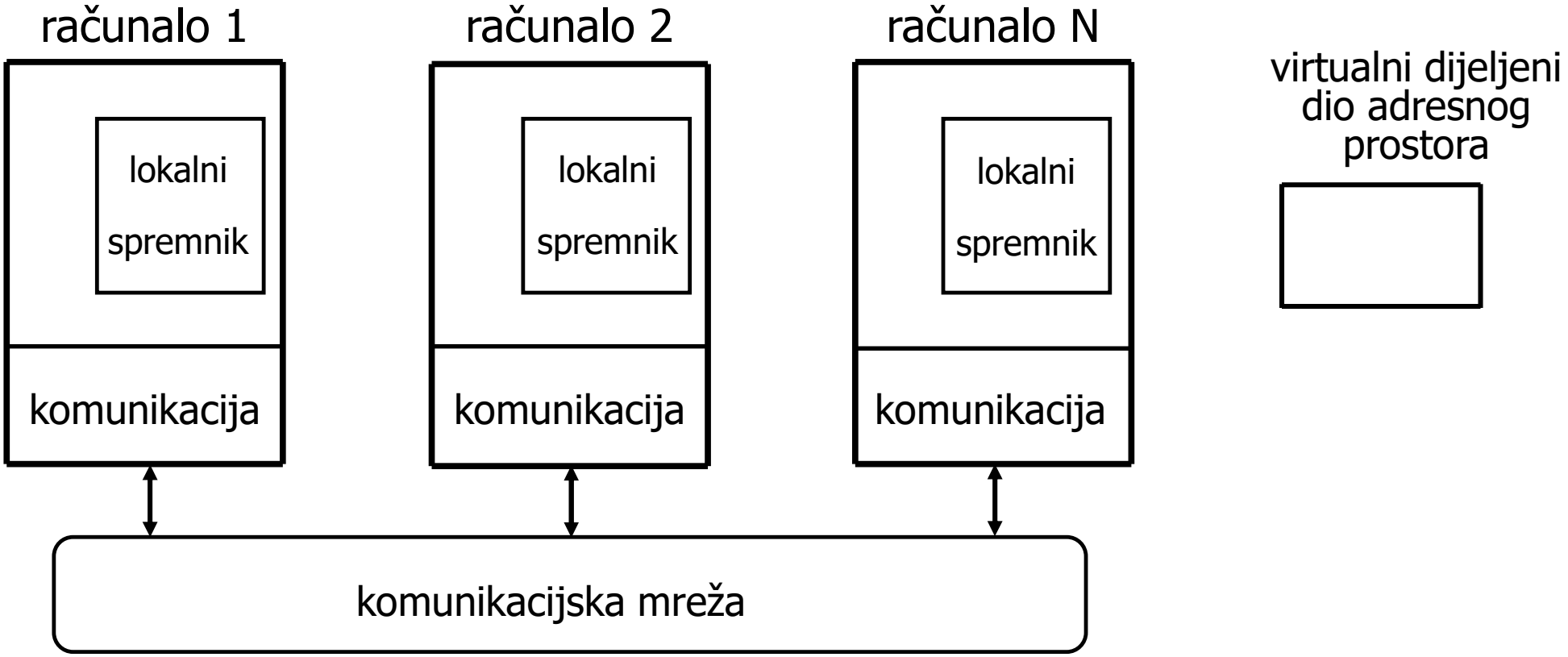
- protokol treba blokirati izvođenje dretve procesa klijenta koja je pozvala udaljenu proceduru dok se ne vrate razultati (primjerice izgradnjom monitora)

Raspodijeljeni dijeljeni spremnički prostor



Mi želimo da jedan dio adresnog prostora dijele svi ti procesi.

- dijeljeni dio prostora može biti samo virtualan, jer procesi mogu stvarno komunicirati samo razmjenom poruka



- kako bi se povećala djelotvornost sustava poželjno bi bilo da se kopije pojedinih stranica koje se često upotrebljavaju istodobno nalaze u više računala
- ako se iz tih stranica samo čita, onda nema nikakvih problema
- međutim, ako jedan od procesa piše u takvu stranicu tada treba načiniti promjene u svim njezinim kopijama
- tako dugo dok postoje razlike u sadržajima stranica dijeljeni spremnik nije *koherentan* odnosno *konzistentan* (u uporabi su oba termina)
- postoje protokoli kojima se podržava koherentnost

1.3. Međusobno isključivanje u raspodijeljenim sustavima

10.3.1. Vremensko uređenje u raspodijeljenim sustavima

- promatramo *N* umreženih računala (čvorova)
- neka se u svakom računalu odvija jedan proces *Pi*
- neka poruke stižu onim redom kako su odašiljane (ta pretpostavka ne mora biti ispunjena u stvarnim mrežema!)
- unutar svakog čvora može se mjeriti vrijeme jer u njemu postoji lokalni sat
- sat koji bi vrijedio za cijelu mrežu ne postoji
- **ne zanima nas apsolutno vrijeme već samo vremensko uređenje pojedinih događaja u sustavu**

Lokalni logički sat Ci

- ponaša se kao brojilo: povećava svoju vrijednost nakon svakog karakterističnog događaja unutar procesa *Pi*
- unutar jednog procesa može se jednoznačno utvrditi redoslijed događaja
- događaju *a* pripada vrijednost logičkog sata *Ci (a)*, a događaju *b* vrijednost logičkog sata *Ci (b)*
- ako je $Ci(a) < Ci(b)$ onda se događaj a dogodio prije događaja *b*, što možemo pisati

$$a \rightarrow b,$$

gdje $\rightarrow$ označava relaciju "dogodilo se prije"

- relacija "dogodilo se prije" može se jednoznačno odrediti:
  - za dva događaje *unutar jednog procesa* ili
  - za događaj slanja poruke od strane jednog procesa i primitka te iste poruke u drugom procesu.
- relacija $a \rightarrow b$ vrijedi:
  - ako su a i b događaji unutar jednog procesa i događaj a se zbio prije događaja b ;
  - ako je a događaj slanja poruka od strane jednog procesa i b događaj primitka te iste poruke od strane drugog procesa.
- relacija je tranzitivna: $(a \rightarrow b) \wedge (b \rightarrow c) \Rightarrow (a \rightarrow c)$
- ako ni $(a \rightarrow b)$ ni $(b \rightarrow a)$ nije istinito, onda događaji a i b nisu vremenski uređeni

Globalni logički sat

Pravila:

- proces *Pi* povećava svoj logički sat *Ci* nakon svakog događaja
- **u našem slučaju događaj je primitak poruke**
- kada proces *Pi* šalje poruku *m* on uz nju pridodaje vremensku oznaku *Tm = Ci*
- kada proces *Pj* primi poruku on postavlja u svoj logički sat *Cj* vrijednost koja je veća od njegove prethodne vrijednosti i veća od prispjele vremenske oznake *Tm:* $Cj = \max\{Cj, Tm\} + 1$

Može se dogoditi:

- da dva lokalna sata *Ci* i *Cj* imaju jednake vrijednosti i
- da se njihove poruke koje si međusobno šalju na putu presretnu noseći jednake vrijednost vremenske oznake

U tom će se slučaju satovi *Ci* i *Cj* postaviti u isto vrijeme.

- tada redoslijed može odlučiti vrijednost indeksa koji su unaprijed dodijeljeni čvorovima
- u sustavu se može podržavati globalni sat koji čine nakupine lokalnih satova

- relacija $a \Rightarrow b$ (a "prethodi" b),

  za događaj a iz procesa Pi i događaj b iz procesa Pj je zadovoljena kada je

$$(Ci(a) < Cj(b))$$

ili

$$(Ci(a) = Cj(b)) \wedge (i < j) .$$

na isti način kao i kod Lamportovog protokola (odjeljak 4.4)

Lamportov algoritam - ponavljanje

- L. Lamport (1974)
- pekarski algoritam (engl. *Lamport's backery algorithm*)

```
dok je (1) {
    ULAZ[I] = 1;
    BROJ[I] = ZADNJI_BROJ + 1;
    ZADNJI_BROJ = BROJ[I];
    ULAZ[I] = 0;
    za (J = 0, J < N, J++){
        dok je (ULAZ[J] == 1);
        dok je ((BROJ[J] != 0) && ((BROJ[J],J) < (BROJ[I],I));
    }
    K.O.;
    BROJ[I] = 0;
    N.K.O.;
}
```

10.3.2. Međusobno isključivanje u raspodijeljenom sustavu

Ponovimo: Međusobno isključivanje jezgrinih funkcija jednog računala

- programski: Dekkerov, Petersonov, Lamportov algoritam
- sklopovska potpora:
  - u jednoprocerskim sustavima međusobno isključivanje moguće postići zabranom prekidanja
  - u višeprocesorskim sustavima potpora isključivanju su instrukcije:TAS, SWAP, FATCH_AND_ADD
- jezgrine funkcije za ostvarenje semafora
  - binarni semafor - u popisu API funkcija taj se semafor obično naziva MUTEX (od engl. mutual exclusion)
  - opći semafor
- jezgrine funkcije za ostvarenje monitora

Međusobno isključivanje u raspodijeljenom sustavu

- na razini mreže ne može se osmisliti nikakva sklopovska potpora za osiguranje međusobnog isključivanja
- u sustavu se samo mogu razmjenjivati poruke
- to je stanje donekle slično onom u višeprocesorskom sustavu u kojem nema nedjeljivog čitanja i pisanja i za koji smo ustanovili uspješnost Lamportova protokola
- treba pokušati oponašati Lamportov protokol u raspodijeljenom okruženju

10.3.3. Protokoli međusobnog isključivanja u raspodijeljenim sustavima

Centralizirani protokol

- jedan od čvorova u mreži može se proglasiti odgovornim za ostvarenje međusobnog isključivanja
- u njemu se nalaze svi podaci i ostali čvorovi moraju od tog čvora tražiti dozvolu za ulazak u kritični odsječak
- proces čvora *Pi* (jedna dretva unutar tog procesa) koji želi ući u kritični odsječaka mora poslati poruku `Zahtjev(i)` centralnom čvoru

- centralni čvor će prihvatiti zahtjev i poslati poruku `Odgovor(i)` čvoru *Pi* kada ustanovi da on smije ući u kritični odsječak
- proces *Pi* (odnosno njegova dretva) mora pričekati da dobije odgovor prije nego što uđe u K.O.
- nakon što dretva u čvoru *i* obavi kritični odsječak ona šalje poruku `Izlaz(i)` centralnom čvoru
- centralni čvor organizira red prispjelih zahtjeva i dozvoljava ulazak u kritični odsječak po redu prispjeća nakon što primi poruku `Izlaz(i)`

ili

- centralni čvor posjeduje značku koju će slati čvorovima koji traže ulazak u kritični odsječak
- značka se može poslati samo jednome od čvorova koji je vraća centralnom čvoru u konačnom vremenu
- nedostatak: velika ovisnost o centralnom čvoru (ako se centralni čvor pokvari protokol će u potpunosti zatajiti)

Protokol s putujućom značkom

- centralni čvor nije ni potreban
- jedna značka koja kao poruka ciklički putuje kroz sve čvorove
- kada poruka značke prispije u čvor i proces Pi će:
  - zadržati značku ako želi ući u kritični odsječak te poslati značku sljedećem čvoru tek nakon što ga završi
  - ili poslati odmah značku sljedećem čvoru ako ne želi ulaziti u kritični odsječak
- slično mehanizmu dodjele sabirnice u višeprocesorskim čvrsto povezanom sustavu
  - sabirnica se redom ciklički nudi svim procesorima a pravo pristupa koriste samo oni procesori koji to žele

Lamportov raspodijeljeni protokol

- zasniva se na uvažavanju vremenskog uređenja temeljenog na globalnom logičkom satu
- kada proces *Pi* želi ući u kritični odsječak generirat će poruku

  `zahtjev(i,T(i))`

  gdje je `T(i)` jednaka vrijednosti logičkog sata *Ci* u trenutku slanja poruke i slati je svim ostalim procesima
- unutar svakog procesa nalazi se red poruka u kojem se svi zahtjevi za ulazak u kritični odsječak svrstavaju u skladu s relacijom $\Rightarrow$ vremenskog uređenja u sustavu
- protokol se izvodi u skladu s pet pravila koja se moraju ostvariti kao lokalne operacije pojedinih čvorova

1. Kada proces *Pi* zahtjeva ulazak u kritični odsječak on:
   - stavlja poruku `zahtjev(i,T(i))` u svoj vlastiti red čekanja;
   - šalje tu poruku svim ostalim procesima.

2. Kada proces *Pj* primi poruku `zahtjev(i,T(i))` on:
   - uskladi svoj lokalni sat *Cj* u skladu s pravilima uspostave globalnog sata;
   - stavlja u svoj red čekanja poruku `zahtjev(i,T(i))`
   - šalje poruku `odgovor(i,T(j))` procesu *Pi*, gdje je `T(j)` jednako novoj vrijednosti logičkog sata *Cj*;

3. Proces *Pi* smije ući u kritični odsječak:
   - kada se njegov vlastiti zahtjev nalazi na početku reda i
   - kada je proces *Pi* primio poruke odgovora svih ostalih procesa čije su vremenske oznake veće od **`T(i)`**.

4. Proces *Pi* obavlja izlazak iz kritičnog odsječka tako da:
   - odstrani iz svog reda svoj **`zahtjev(i,T(i))`**
   - pošalje svim ostalim procesima poruku

     `izlazak(i,T(i))`,

     gdje je **`T(i)`** jednaka vrijednosti iz prvotno poslanog zahtjeva.

5. Kada proces *Pj* primi poruku **`izlazak(i,T(i))`** on iz svog reda čekanja odstranjuje zahtjev procesa *Pi*.

Za ulazak i izlazak iz kritičnog odsječka u sustavu se razmijeni *3 x (N - 1)* poruka:

- *(N-1)* poruka zahtjev,
- *(N-1)* poruka odgovor i
- *(N-1)* poruka izlazak.

Protokol Ricarta i Agrawala

- Kako smanjiti broj poruka? Nema poruke **`izlaz`**.
- Procesi šalju odgovore na primljene poruke samo onda ako ustanove da ne žele ulaziti u K.O. ili ustanove da proces koji je postavio zahtjev ima pravo prvenstva.

- Protokol se ostvaruje sa sljedeća četiri pravila:

1. Kada proces Pi zahtjeva ulazak u kritični odsječak on:
   - šalje poruku **`zahtjev(i,T(i))`** svim ostalim procesima (gdje je **`T(i)`** trenutna vrijednost lokalnog sata u čvoru *i*).

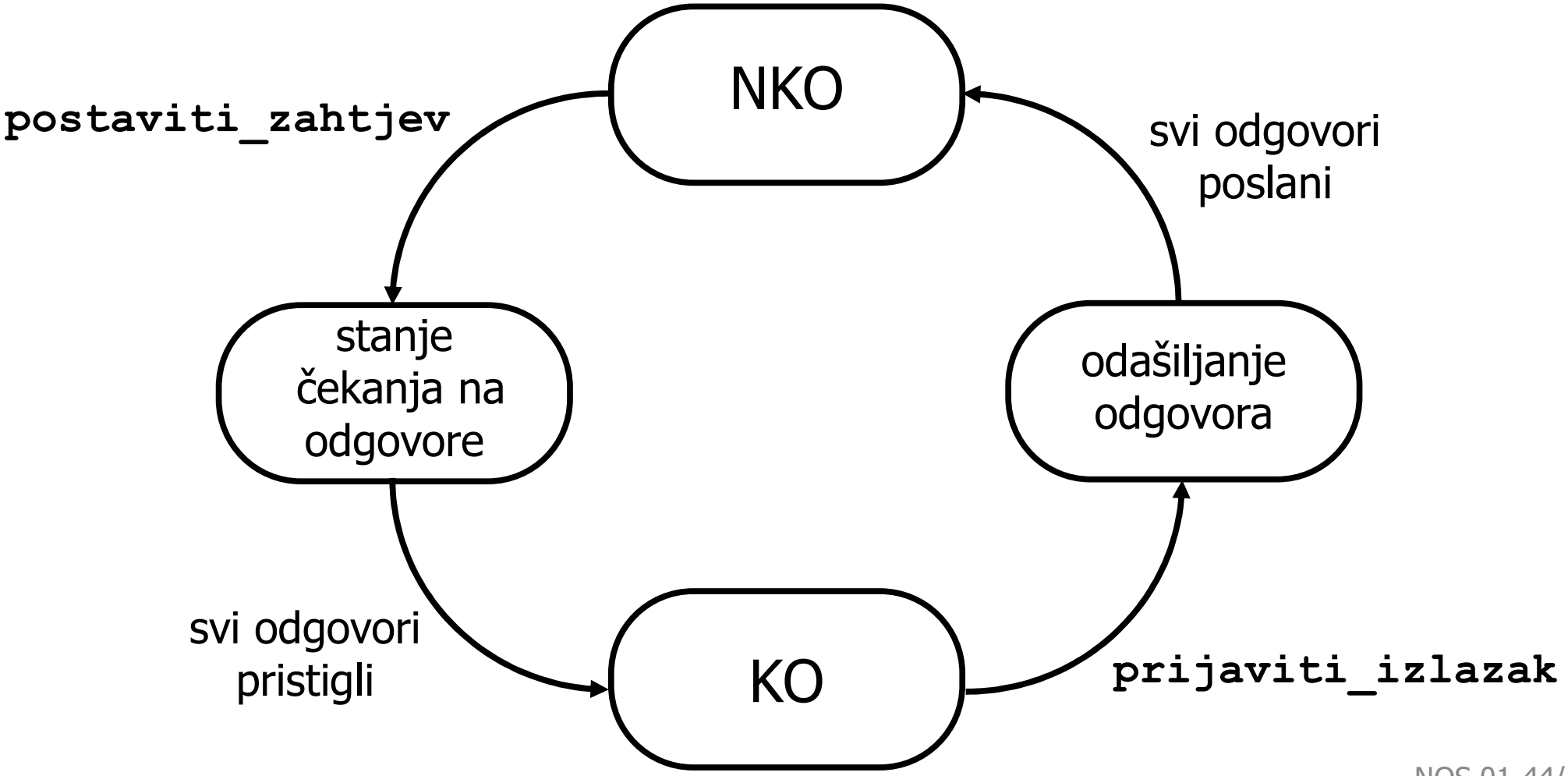
2. Kada proces *Pj* primi poruku `zahtjev(i,T(i))` on:
   - uskladi svoj lokalni sat *Cj* u skladu s pravilima uspostave globalnog sata;
   - šalje poruku `odgovor(j,T(i))` ako ne želi ulaziti u K.O. ili ako je njegov zahtjev za ulazak došao kasnije;
   - proces *Pj* ,dakle, neće poslati odgovor ako postoji njegov zahtjev čija vremenska oznaka `(j,T(j))` prethodi vremenskoj oznaci `(i,T(i))`.
3. Kada proces *Pi* primi odgovore svih ostalih čvorova on smije ući u kritični odsječak.
4. Kada proces *Pi* izlazi iz kritičnog odsječka on će poslati poruku `odgovor(i,T(j))` svim procesima čiji zahtjevi kod njega čekaju na odgovor.

Za ulazak i izlazak iz kritičnog odsječka u sustavu se razmijeni $2 \times (N - 1)$ poruka:

- $(N-1)$ poruka zahtjev i
- $(N-1)$ poruka odgovor.

- Opisana pravila za ostvarenje protokola trebalo bi pretvoriti u lokalne funkcije svakog čvora koje se obavljaju nedjeljivo kako je to prikazano u primjeru 10.4. u udžbeniku gdje sinkronizacija ostvarena uz pomoć monitora.

Stanja dretve u postupku sinkronizacije s drugim dretvama u raspodijeljenom okruženju



Analiza protokola Ricarta i Agrawala

- Svi su zahtjevi vremenski uređeni jer:
  - svakom se novom zahtjevu pripisuje lokalno vrijeme `lokalni_sat` čija je vrijednost veća od vrijednosti viđene u od bilo kojeg zahtjeva koji je prije toga stigao u čvor
  - problem jednakih brojeva koji se mogu pojaviti zbog istovremenog generiranja zahtjeva u različitim čvorovima razrješavaju se indeksom čvora (u skladu s pravilima logičkog sata).
- Odlučivanje je potpuno raspodijeljeno.
- Ne može se pojaviti potpuni zastoj.
- Izgladnjivanje također nije moguće.
- Algoritam se obavlja razmjenom $2 \times (N - 1)$ poruka.